



Optimal Tiled QR Factorization

Jeffrey Larson
KTH Automatic Control Group
JANUARY 8, 2014

Motivation

- The current computer architecture has moved towards the multi/many-core structure.
- However, the algorithms in the current sequential dense numerical linear algebra libraries (e.g. LAPACK) do not parallelize/scale well on multi/many-core architectures.
- A new family of algorithms, the *tilted algorithms*, has recently been introduced to circumvent this problem.

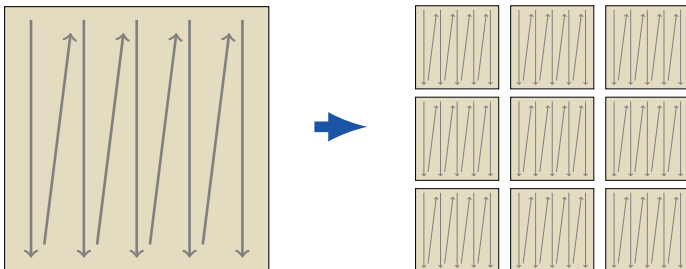
Motivation

Tiled algorithms seek to alleviate these by

- using asynchronous scheduling of tasks to remove synchronization points
- breaking up the larger tasks to allow for fine-grained parallelism
- enabling better (optimal) memory-to-computation ratio (“communication avoiding”)

Tiled Algorithms - Data Layout

Reordering the data of the matrix into smaller regions of contiguous memory enables having the data layout match the algorithm.



Thus an $n \times n$ matrix is made up of t tiles of size $n_b \times n_b$ so that $n = t \cdot n_b$. The tile size becomes a tuning parameter and can depend upon the machine as well as the algorithm.

Tiled QR Factorization

A QR factorization of a rectangular matrix A is a decomposition of A as

$$A = QR$$

where Q is a unitary matrix (i.e. $Q^T Q = I$) and R is an upper triangular matrix. If A is nonsingular, then this factorization is unique when R is required to have positive diagonal entries.

Tiled QR Factorization

A QR factorization of a rectangular matrix A is a decomposition of A as

$$A = QR$$

where Q is a unitary matrix (i.e. $Q^T Q = I$) and R is an upper triangular matrix. If A is nonsingular, then this factorization is unique when R is required to have positive diagonal entries.

MATLAB uses this to solve

$$Ax = b$$

Tiled QR Factorization

A QR factorization of a rectangular matrix A is a decomposition of A as

$$A = QR$$

where Q is a unitary matrix (i.e. $Q^T Q = I$) and R is an upper triangular matrix. If A is nonsingular, then this factorization is unique when R is required to have positive diagonal entries.

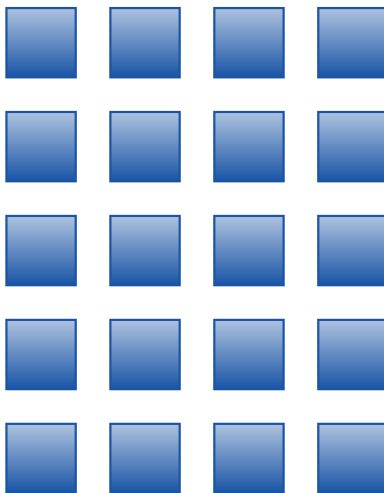
MATLAB uses this to solve

$$Ax = b$$

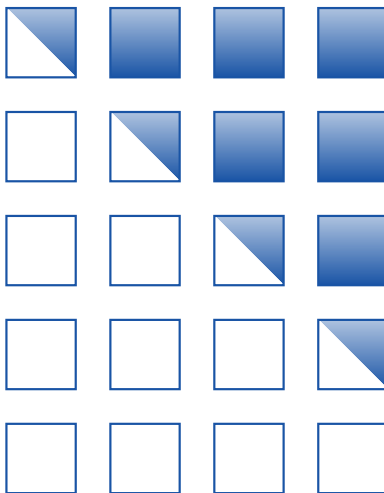
$$QRx = b \implies Rx = Q^T b$$

Solve this using back substitution.

Goal



Goal



Tools

code name

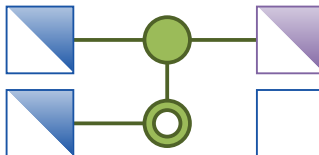
weight

GEQRT




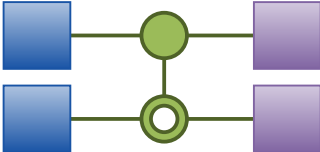
4

TTQRT

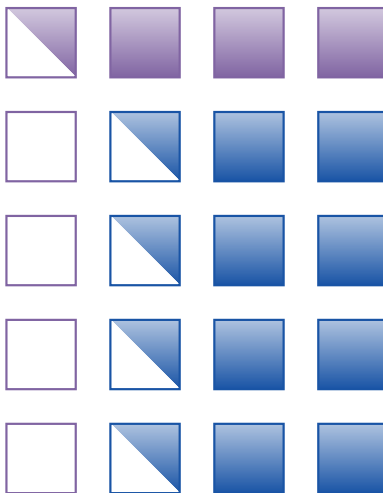


2

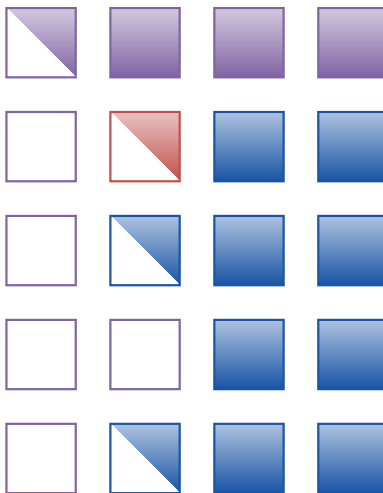
Updates

code name		weight
UNMQR		6
TTMQR		6

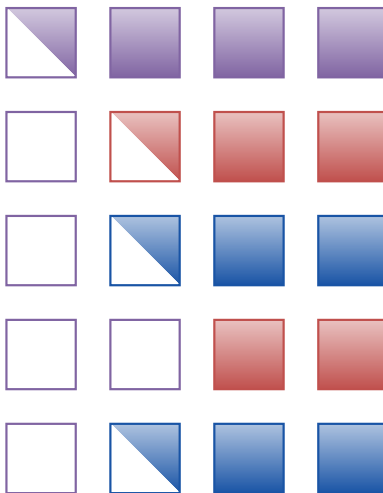
Example



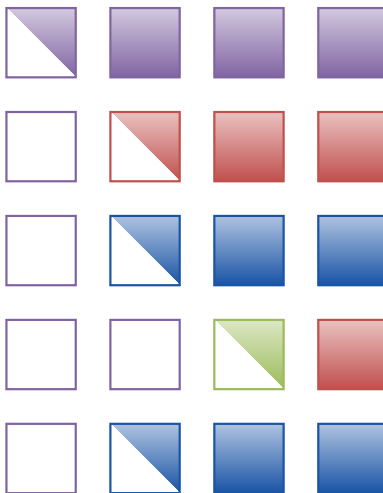
Example



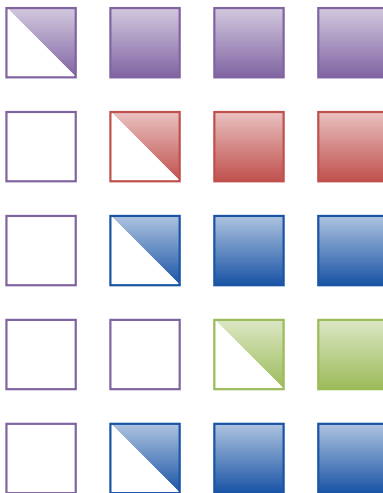
Example



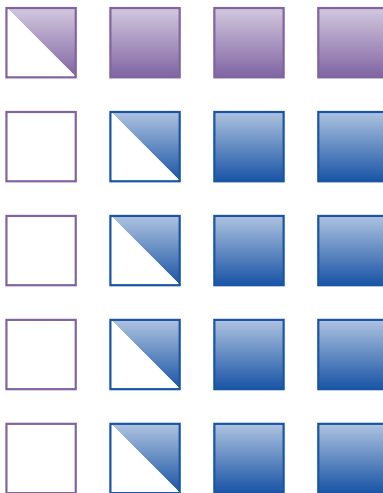
Example



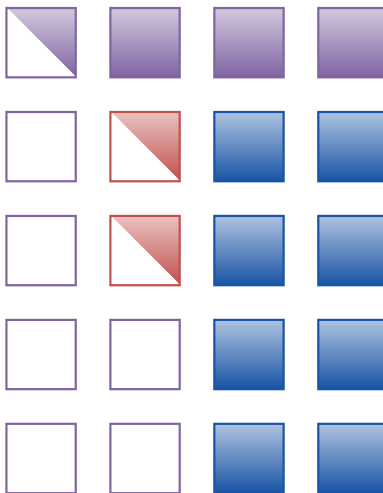
Example



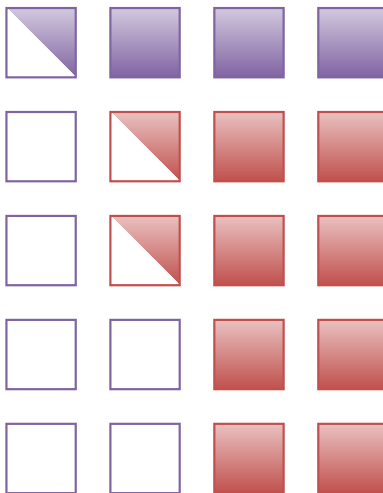
Example



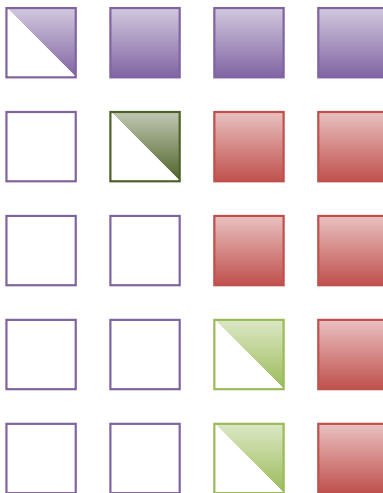
Example



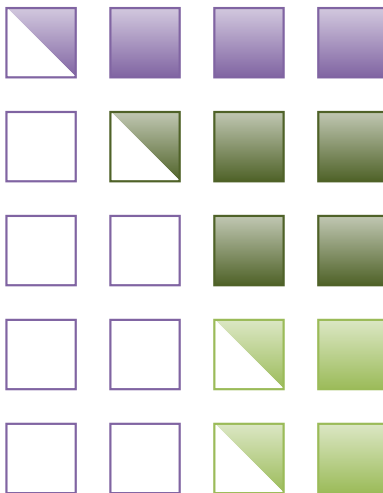
Example



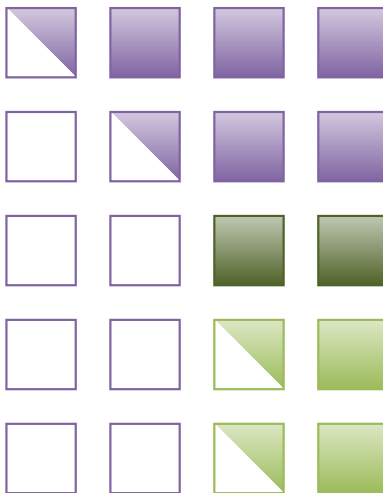
Example



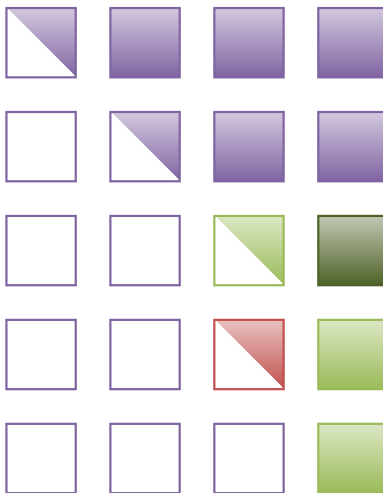
Example



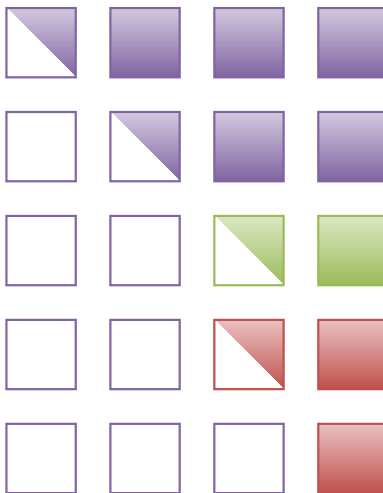
Example



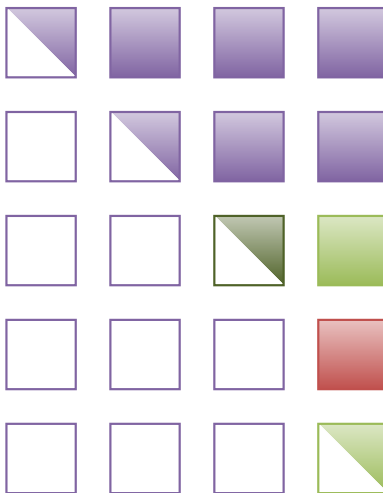
Example



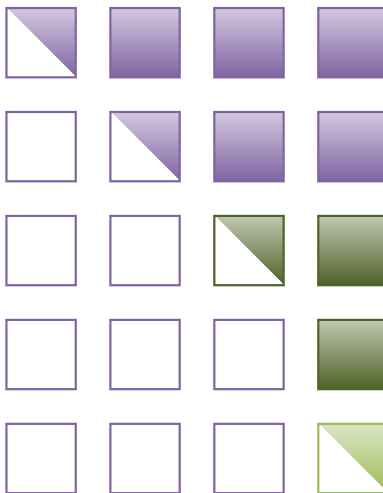
Example



Example

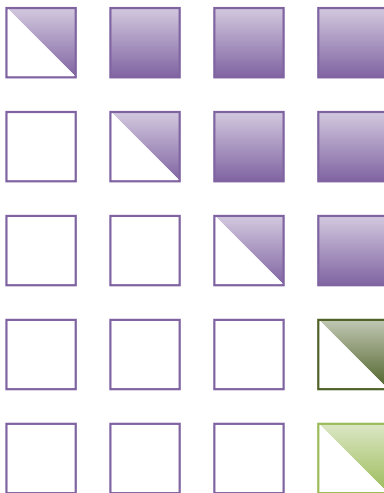


Example

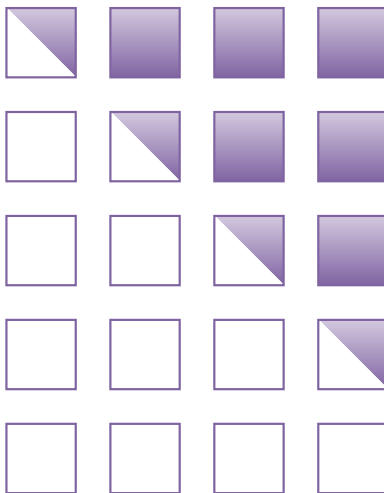




Example



Example



Rules

- Every tile below the diagonal must become a zero.
- Every tile on the diagonal must become a triangle.
- Every triangularization or zeroing requires an update.
- Updates must be applied in order.
- Only a triangle can zero another triangle.
- Before triangularizing, tiles to the left must be zeros.

Model Variables

Update after triangularization

$w_{ikl} = t \in [0, T] :$ if we finish the update of tile (i, k) at time t
(This update was necessitated by $x_{il} = s : l < k, s < t$).

Triangularization

$x_{ik} = t \in [0, T] :$ if we complete triangularization of tile (i, k) at time t

Update after zeroing

$y_{ijkl} = t \in [0, T] :$ if we finish the update of tiles (i, k) and (j, k) at t
(This update was necessitated by $z_{ijl} = s : l < k, s < t$)

Zeroing

$z_{ijk} = t \in [0, T] :$ if we complete zeroing tile (i, k) using tile (j, k) at t

Example Constraints

After a tile is triangularized, updates must occur in the next columns.

$$x_{ik} \leq w_{ilk} - 3 \quad \forall i, k < q, i \geq k, l > k$$

Example Constraints

After a tile is triangularized, updates must occur in the next columns.

$$x_{ik} \leq w_{ilk} - 3 \quad \forall i, k < q, i \geq k, l > k$$

Tiles strictly below the diagonal must be zeroed at some point.

$$\sum_j \hat{z}_{ijk} = 1 \quad \forall i > k$$

Example Constraints

After a tile is triangularized, updates must occur in the next columns.

$$x_{ik} \leq w_{ilk} - 3 \quad \forall i, k < q, i \geq k, l > k$$

Tiles strictly below the diagonal must be zeroed at some point.

$$\sum_j \hat{z}_{ijk} = 1 \quad \forall i > k$$

After a tile (i, k) is zeroed, we can't use it to zero.

$$z_{ijk} \geq z_{hik} \quad \forall h, i, j, k$$

Known Results

- Greedy is not optimal!
- If the number of processors is unlimited, then an optimal procedure is known.
- If the number of processors is limited, then the optimal routine is not known.

Optimal vs. Existing

If the matrix is divided into 6 rows and 3 columns of tiles, and 6 processors are available, the optimal schedule is better than any known method. Also, the zeroing pattern does not match any known algorithms for Tiled QR.

	1	2	3
1	*		
2	1	*	
3	1	2	*
4	1	2	3
5	1	2	3
6	2	4	4

(a) Optimal

	1	2	3
1	*		
2	1	*	
3	1	2	*
4	1	2	3
5	1	2	3
6	1	2	3

(b) Flat Tree

	1	2	3
1	*		
2	1	*	
3	2	2	*
4	1	3	3
5	2	3	4
6	3	4	5

(c) Greedy

So where is this useful?

- Solving the IP model takes quite a bit of time.

So where is this useful?

- Solving the IP model takes quite a bit of time.
- We have a collection of matrices where optimal is better than any existing method.
- For example: the optimal for $6 \times 3 - 6$ takes 7% less time than any known method.

So where is this useful?

- Solving the IP model takes quite a bit of time.
- We have a collection of matrices where optimal is better than any existing method.
- For example: the optimal for $6 \times 3 - 6$ takes 7% less time than any known method.
- Many image processing examples involve repeatedly factoring matrices that are the same size.

Why does this matter?